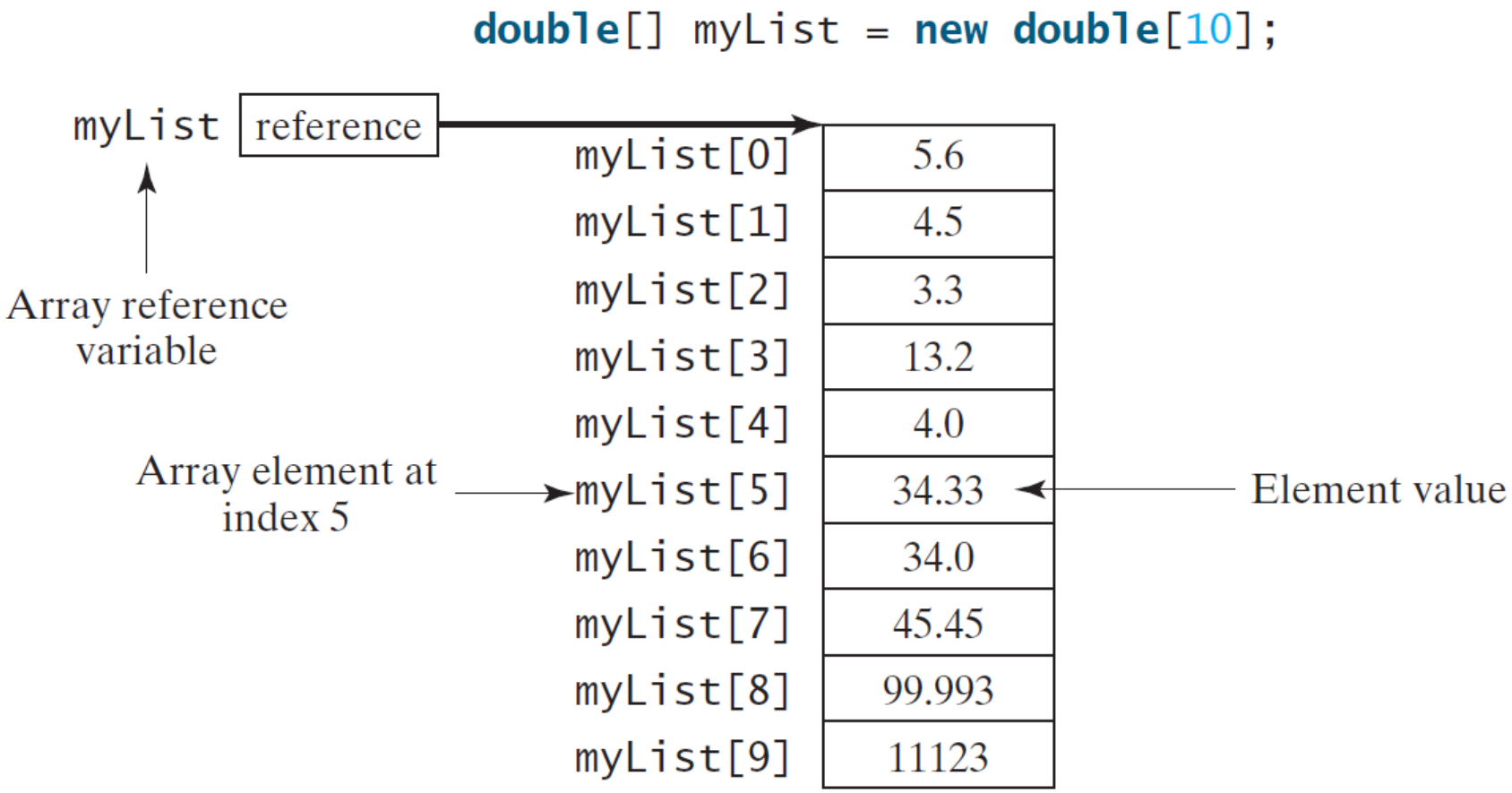
جامعة بيرزيت
BIRZEIT UNIVERSITY

Arrays

Introducing Arrays

❖ Array is a data structure that represents a collection of the **same** types of data.



Declaring Array Variables

datatype[] arrayRefVar;

Example:

double[] myList;

datatype arrayRefVar**[]**; **// This style is allowed, but not preferred**

Example:

double myList[];

Creating Arrays

arrayRefVar = new datatype[arraySize];

Example:

myList = new double[10];

- **myList[0]** references the 1st element in the array.
- **myList[9]** references the last element in the array.

Declaring and Creating in 1 Step

```
datatype[] arrayRefVar = new datatype[arraySize];
```

```
double[] myList = new double[10];
```

```
datatype arrayRefVar[] = new datatype[arraySize];
```

```
double myList[] = new double[10];
```

The Length of an Array

- Once an array is created, its **size is fixed**.
- It cannot be changed.
- You can find its size using:

arrayRefVar.length

For example:

myList.length → returns 10

Default Values

❖ When an array is created, its elements are assigned the **default value** of :

- **0** for the numeric data types.
- '**\u0000**' for **char** types.
- **false** for **boolean** types.

Indexed Variables

❖ The array elements are accessed through the **index**.

❖ The array indices are ***0-based***, i.e., it starts from **0** to **arrayRefVar.length-1**.

❖ Each element in the array is represented using the following syntax, known as an *indexed variable*:

arrayRefVar[index];

Using Indexed Variables

❖ After an array is created, an indexed variable can be used in the same way as a regular variable.

❖ For example, the following code adds the value in **myList[0]** and **myList[1]** to **myList[2]**:

```
myList[2] = myList[0] + myList[1];
```

Array Initializers

- Declaring, creating, initializing in 1 step:

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

- This shorthand notation is equivalent to the following statements:

```
double[] myList = new double[4];

myList[0] = 1.9;

myList[1] = 2.9;

myList[2] = 3.4;

myList[3] = 3.5;
```

Trace Program with Arrays

```
public class Test {
  public static void main(String[] args) {
      int[] values = new int[5];
      for (int i = 1; i < 5; i++) {
          values[i] = i + values[i-1];
      }
      values[0] = values[1] + values[4];
  }
}
```

Initializing arrays with input values

```
Scanner input = new Scanner(System.in);

System.out.print("Enter "  +  myList.length  +  " values: ");

for (int i = 0 ;  i < myList.length  ;   i++)

        myList[ i ] = input.nextDouble();
```

Initializing arrays with random values

```
for (int i = 0; i < myList.length; i++)
    myList[i] = Math.random() * 100;
```

Printing arrays

```
for (int i = 0; i < myList.length; i++)
    System.out.print(myList[i] + " ");
```

Summing all elements

```
double total = 0;
for (int i = 0; i < myList.length; i++)
    total += myList[i];
```

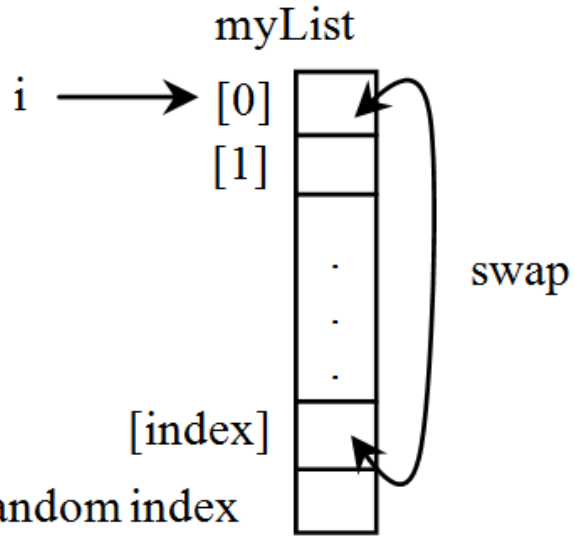
Finding the largest element

```
double max = myList[0];
for (int i = 1; i < myList.length; i++) {
    if (myList[i] > max)
        max = myList[i];
}
```

Random Shuffling

```
for (int i = 0; i < myList.length; i++) {
  // Generate an index j randomly
  int index = (int)(Math.random()
    * myList.length);

  // Swap myList[i] with myList[index]
  double temp = myList[i];
  myList[i] = myList[index];
  myList[index] = temp;
}
```

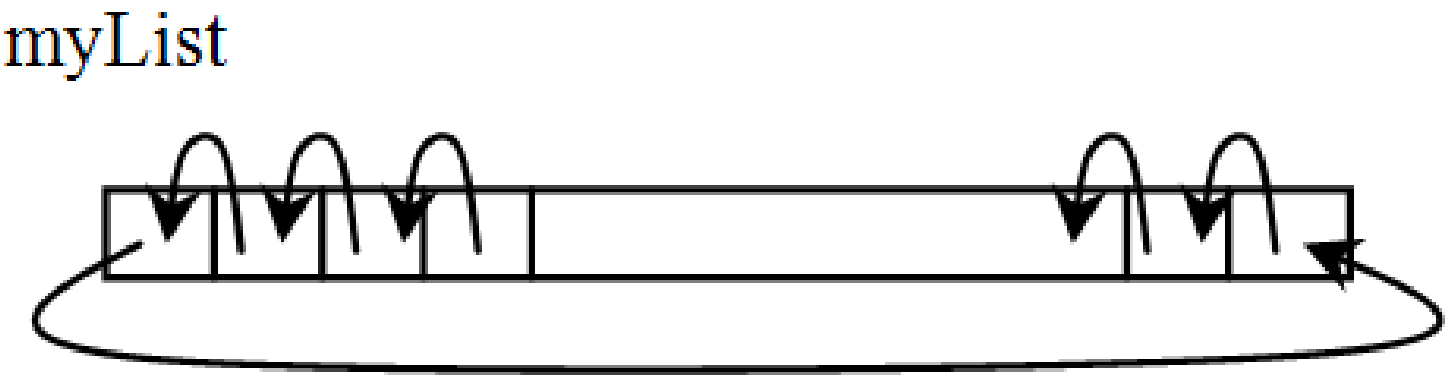


Shifting Elements

```
double temp = myList[0]; // Retain the first element

// Shift elements left
for (int i = 1; i < myList.length; i++) {
  myList[i - 1] = myList[i];
}

// Move the first element to fill in the last position
myList[myList.length - 1] = temp;
```



Enhanced for Loop (for-each loop)

❖ **JDK 1.5** introduced a new for loop that enables you to traverse the complete array sequentially without using an index variable.

❖ For example, the following code displays all elements in the array **myList**:

```
for (double value: myList)   System.out.println(value);
```

❖ In general, the syntax is:

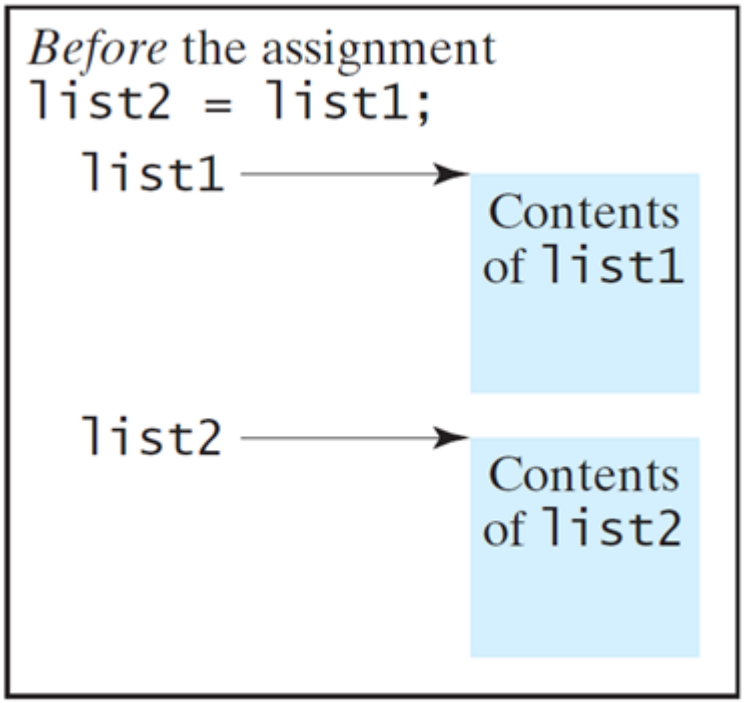
```
for (elementType value: arrayRefVar) {
    // Process the value
}
```

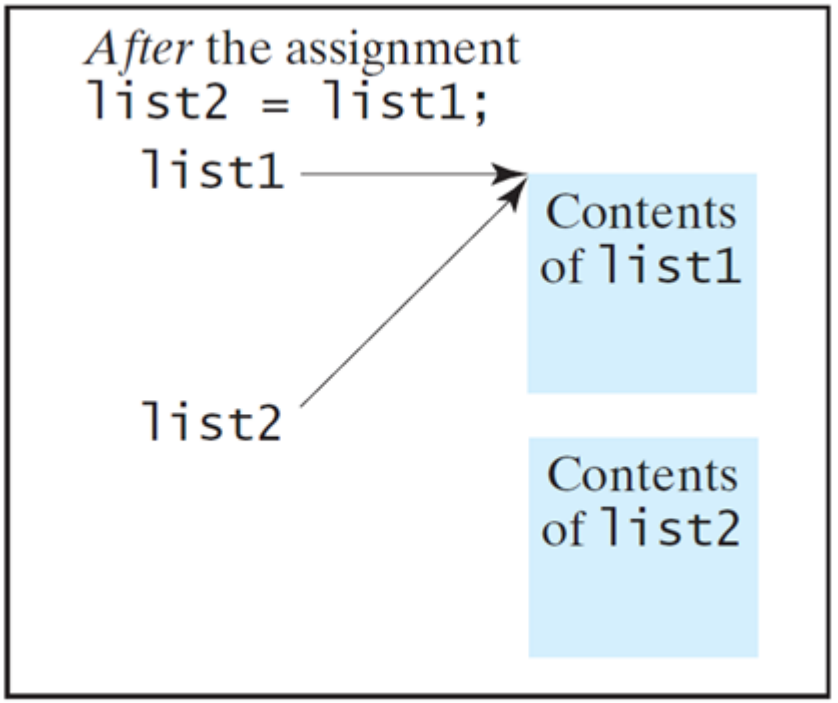
❖ You still have to use an index variable if you wish to traverse the array in a different order or change the elements in the array.

Copying Arrays

❖ Often, in a program, you need to duplicate an array or a part of an array. In such cases you could attempt to use the assignment statement (=), as follows:

list2 = list1;





Copying Arrays

- Using a loop:

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new int[sourceArray.length];

for (int i = 0; i < sourceArrays.length; i++)
    targetArray[i] = sourceArray[i];
```

The arraycopy Utility

System.arraycopy(sourceArray, src_pos, targetArray, tar_pos, length);

❖ Example:

System.arraycopy(sourceArray, 0, targetArray, 0, sourceArray.length);

Passing Arrays to Methods

```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
  }
}
```

- **Invoke the method**

```
int[] list = {3, 1, 2, 6, 4, 2};
printArray(list);
```

Anonymous Array

- The statement

  printArray(**new int[]{3, 1, 2, 6, 4, 2}**);

- Creates array using the following syntax:

  new dataType[]{literal0, literal1, ..., literalk}

- There is no explicit reference variable for the array.

- Such array is called an ***anonymous array***.

Pass by Value

❖ For a parameter of a **primitive type value**, **the actual value is passed**.

- Changing the value of the local parameter inside the method **does not affect** the value of the variable outside the method.

❖ For a parameter of an **array type**, the value of the parameter contains a reference to an array; **this reference is passed to the method**.

- Any changes to the array that occur inside the method body **will affect** the original array that was passed as the argument.

Simple Example

```
public class Test {
  public static void main(String[] args) {
    int x = 1;
    int[] y = new int[10];

    m(x, y);

    System.out.println("x is " + x);
    System.out.println("y[0] is " + y[0]);
  }

  public static void m(int number, int[] numbers) {
    number = 1001;
    numbers[0] = 5555;
  }
}
```

Returning an Array from a Method

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];
  for (int i=0, j=result.length - 1; i < list.length/2; i++, j--) {
        result[j] = list[i];
  }
  return result;
}
```

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

Linear Search

❖ The linear search approach compares the key element, **key**, ***sequentially*** with each element in the array **list**.

❖ The method continues to do so until the key matches an element in the list or the list is exhausted without a match being found.

❖ If a match is made, the linear search returns the **index** of the element in the array that matches the key.

❖ If no match is found, the search returns **-1**.

From Idea to Solution

```
public static int linearSearch(int[] list, int key) {
  for (int i = 0; i < list.length; i++)
      if (key == list[i])     return i;
  return -1;
}
```

Trace the method:

```
int[] list = {1, 4, 4, 2, 5, -3, 6, 2};
int i = linearSearch(list, 4);  // returns 1
int j = linearSearch(list, -4); // returns -1
int k = linearSearch(list, -3); // returns 5
```

The Arrays.binarySearch Method

- Since binary search is frequently used in programming, Java provides several **binarySearch** methods for searching a key in an array of int, double, char, short, long, and float in the **java.util.Arrays** class.

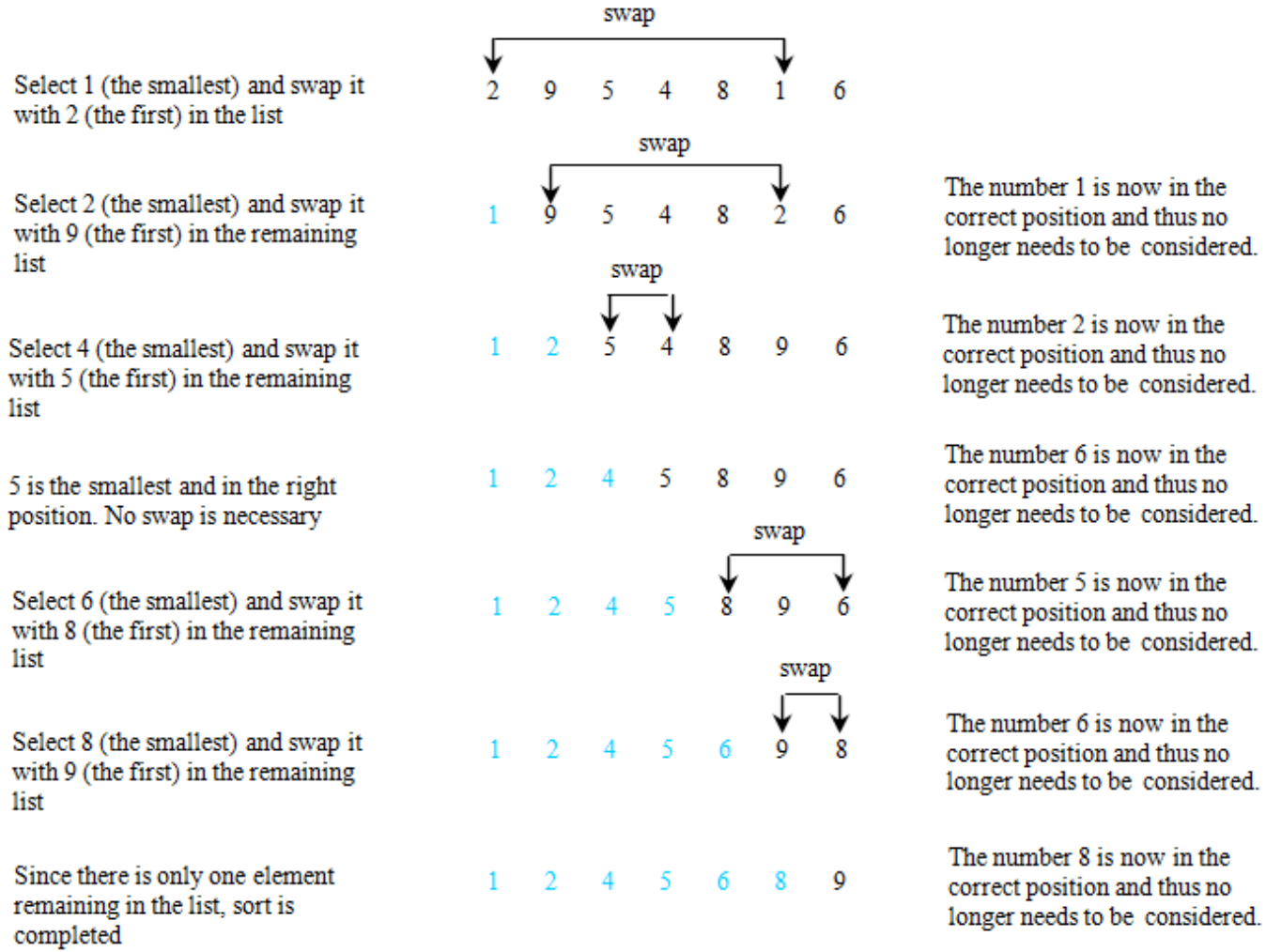
```
int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};
System.out.println("Index is " +  Arrays.binarySearch(list, 11));


char[] chars = {'a', 'c', 'g', 'x', 'y', 'z'};
System.out.println("Index is " + Arrays.binarySearch(chars, 't'));
```

- For the **binarySearch** method to work, the array must be pre-sorted in increasing order.

Selection Sort

❖ Selection sort finds the smallest number in the list and places it first. It then finds the smallest number remaining and places it second, and so on until the list contains only a single number.



From Idea to Solution

```
for (int i = 0; i < list.length; i++)   {
  select the smallest element in list[i..listSize-1];
  swap the smallest with list[i], if necessary;
  // list[i] is in its correct position.
  // The next iteration apply on list[i+1..listSize-1]
}
```

The Arrays.sort Method

- Java provides several sort methods for sorting an array of **int**, **double**, **char**, **short**, **long**, and **float** in the **java.util.Arrays** class.
- For example, the following code sorts an array of numbers and an array of characters:

```
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};
java.util.Arrays.sort(numbers);

char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};
java.util.Arrays.sort(chars);
```

main Method is just a Regular Method

- You can call a regular method by passing actual parameters.
- You can pass **arguments** to **main**.
- For example, the main method in class **B** is invoked by a method in **A**, as shown below:

```
class B {
  public static void main(String[] args) {
    for (int i = 0; i < args.length; i++)
      System.out.println(args[i]);
  }
}
```

```
public class A {
  public static void main(String[] args) {
    String[] strings = {"New York",
      "Boston", "Atlanta"};
    B.main(strings);
  }
}
```

Command-Line Parameters

```
class TestMain {
    public static void main(String[] s) {
        ...
    }
}
```

```
java TestMain arg0 arg1 arg2 ... argn
```

❖ In the **main** method, get the arguments from **s[0], s[1], ..., s[n]**, which corresponds to **arg0, arg1, ..., argn** in the command line.

Problem: Calculator

- Objective: Write a program that will perform binary operations on integers. The program receives three parameters: an operator and two integers.

```
java Calculator 2 + 3
java Calculator 2 - 3
java Calculator 2 / 3
java Calculator 2 . 3
```

Declare/Create 2D Arrays

```
// Declare array refvar
dataType[][] refVar;

// Create array and assign its reference to variable
refVar = new dataType[10][10];

// Combine declaration and creation in one statement
dataType[][] refVar = new dataType[10][10];

// Alternative syntax
dataType refVar[][] = new dataType[10][10];
```

Creating 2D Arrays

```
int[][] matrix = new int[10][10];

for (int i = 0; i < matrix.length; i++)
  for (int j = 0; j < matrix[i].length; j++)
    matrix[i][j] = (int)(Math.random() * 1000);
```

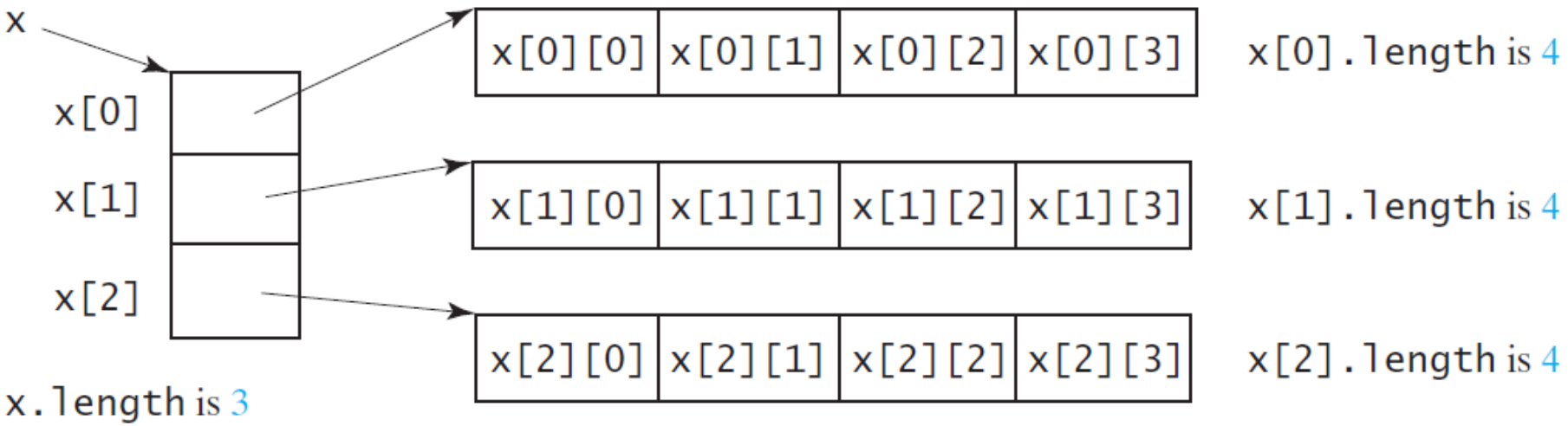
Declaring, Creating, and Initializing Using Shorthand Notations

- You can also use an array initializer to declare, create and initialize a 2-dimensional array.
- For example:

```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

Lengths of 2D Arrays

```
int[][] x = new int[3][4];
```



Lengths of 2D Arrays, cont.

```
int[][] array = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9},
        {10, 11, 12}
};
```

array.length

array[0].length

array[1].length

array[2].length

array[3].length

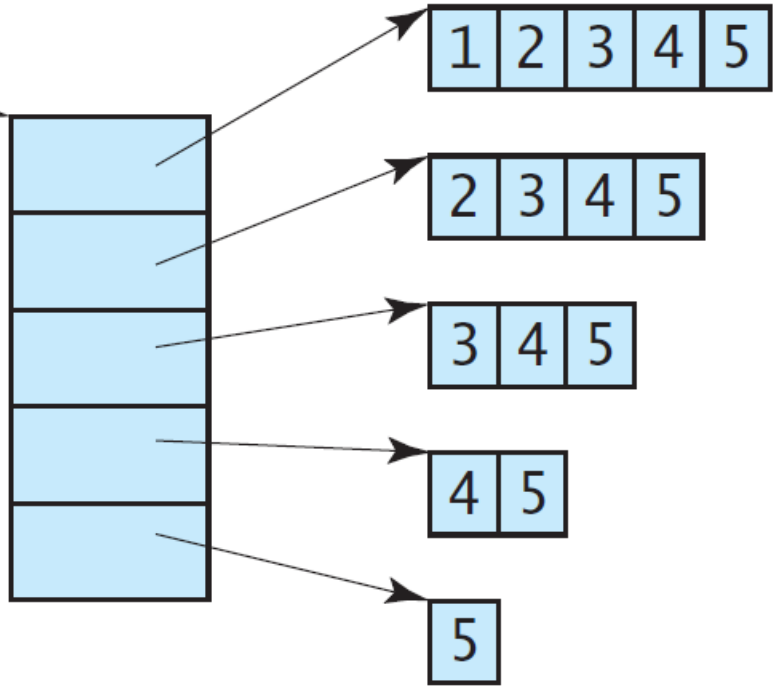
array[4].length → **ArrayIndexOutOfBoundsException**

Ragged Arrays

- Each row in a 2D array is **itself** an array. So, the **rows can have different lengths**.
- Such an array is known as *a **ragged array***.

For example:

```
int[][] triangleArray = {
  {1, 2, 3, 4, 5},
  {2, 3, 4, 5},
  {3, 4, 5},
  {4, 5},
  {5}
};
```



Printing arrays

```
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length;
                    column++) {
        System.out.print(matrix[row][column] + " ");
  }

  System.out.println();
}
```

What is Sudoku?

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6							
			4	1	9			5
				8			7	9

Checking Whether a Solution Is Correct

5	3	*4*	*6*	7	*8*	*9*	*1*	*2*
6	*7*	*2*	1	9	5	*3*	*4*	*8*
1	9	8	*3*	*4*	*2*	*5*	6	*7*
8	*5*	*9*	*7*	6	*1*	*4*	*2*	3
4	*2*	*6*	8	*5*	3	*7*	*9*	1
7	*1*	*3*	*9*	2	*4*	*8*	*5*	6
9	6	*1*	*5*	*3*	*7*	*2*	*8*	*4*
2	*8*	*7*	4	1	9	*6*	*3*	5
3	*4*	*5*	*2*	8	*6*	*1*	7	9

Multidimensional Arrays

❖ Occasionally, you will need to represent

n-dimensional data structures.

❖ In Java, you can create n-dimensional arrays for any integer n.

❖ The way to declare two-dimensional array variables and create two-dimensional arrays can be generalized to declare n-dimensional array variables and create n-dimensional arrays for **n > 2**.

Multidimensional Arrays

```
double[][][] scores = {
  {{7.5, 20.5}, {9.0, 22.5}, {15, 33.5}, {13, 21.5}, {15, 2.5}},
  {{4.5, 21.5}, {9.0, 22.5}, {15, 34.5}, {12, 20.5}, {14, 9.5}},
  {{6.5, 30.5}, {9.4, 10.5}, {11, 33.5}, {11, 23.5}, {10, 2.5}},
  {{6.5, 23.5}, {9.4, 32.5}, {13, 34.5}, {11, 20.5}, {16, 7.5}},
  {{8.5, 26.5}, {9.4, 52.5}, {13, 36.5}, {13, 24.5}, {16, 2.5}},
  {{9.5, 20.5}, {9.4, 42.5}, {13, 31.5}, {12, 20.5}, {16, 6.5}}};
```